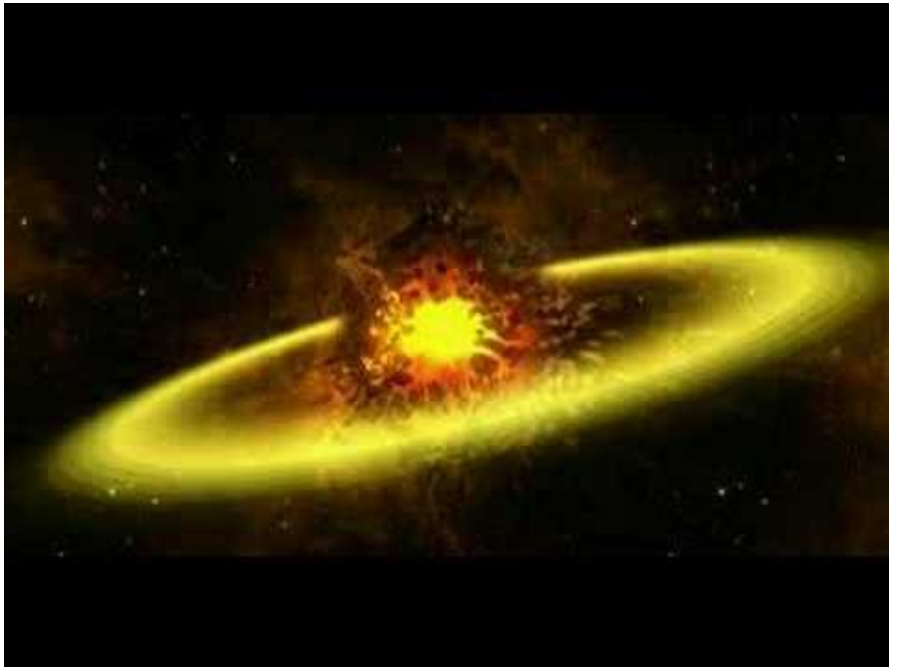
PostgreSQL and backups
With a differential touch

Michael Paquier
PGConf Russia 2016
2015/02/03~2015/02/05, Moscow

Why backups?



In short

- Backups
  - Methods
  - Planning
  - Retention
  - Performance
- Restore
  - Methods
  - QE/QA
  - Performance

About backups

- Replication
- Logical backups
  - pg_dump
  - pg_dumpall
- Physical backups
  - pg_basebackup
  - FS-level
  - Etc.

Replication

- Live backups
- Fast
- Shortest restore time
- Those are not real backups!
- Delayed standbys leverage that.
  - recovery_min_apply_delay in recovery.conf
  - Delays transaction commits at replay

Logical backups

- pg_dump
  - Use -Fc, custom format
- Compression with zlib
  - After 3~5, usually no real difference
  - Higher = More CPU
  - Lower = More write I/O
- Other things
  - Object-level granularity
  - --jobs for parallel dump
- **pg_dumpall -g**

Logical - performance

- Single transaction
- Single backend – 1 CPU per each
- Fine for up to 100GB (still big)
- Postgres cache is fine
- Disk
  - Throttled by disk I/O
  - Better on separate disk than PGDATA

Logical - Restore

- CREATE INDEX can be costly
  - Large speedup in 9.5
- Accelerate things
  - fsync = off
  - wal_level = minimal
  - archive_mode = off
- If fsync = off
  - Reenable it after!
  - Drop OS caches

pg_restore

- Use single transaction -1 (less WAL generated)
- DROP DATABASE if crash
- Parallel restore -j
  - Incompatible with -1
  - Better using it in most cases
- Time depends on
  - Data size
  - Schema, objects to rebuild from scratch

Physical backup

- Dump files of the database
- Faster than logical
- Architecture, version and compile-option dependent
- Cluster-level backup only

Physical - Methods

- FS-level snapshots
  - Need to be atomic
  - Including all tablespaces
- pg_basebackup
- Low-level
  - pg_start_backup()
  - Custom method: cp, rsync, tar, FS-snapshot
  - pg_stop_backup()

pg_basebackup

- Configuration
  - wal_level = [archive|hot_standby|logical]
  - max_wal_senders >= 1
- -x to stream enough WAL segments in backup itself
- Complete PGDATA backup
- Can map tablespaces to new location
- Impact
  - Single-threaded
  - Sequential read

WAL archiving

- Configuration
  - archive_mode = on (or 'shared' in 9.5)
  - archive_command = archive segment X
- In recovery
  - restore_command = Get back segment X

WAL archiving - limitations

- Holes in WAL history
  - archive_mode = shared
  - Needs standbys
- pg_receivexlog
  - Does archiving, like a standby for master
  - Synchronous mode in 9.5

Physical - Restore

- PITR
- Time depends on distance to target
  - WAL replay
- Effects
  - Random writes
  - Single threaded (startup process) + alpha

So...

- Backup time may not matter
- Restore time **is** critical
- Test your backups
  - Nothing immediately in production
- Right solution may be to mix all methods
  - Fewer backups, more WAL logs
  - More backups, less WAL segments
  - With pg_dump on top for disasters
- Backup policy
  - Retention
  - Frequency

Differential backups – Why?

- Delta backups
  - Need prior full backup
  - Used to rebuild newer full backups
- Why?
  - More backups = More **full** backups
  - Large data sets impact policy retention
  - Full backup size <=> Time to take it and store it

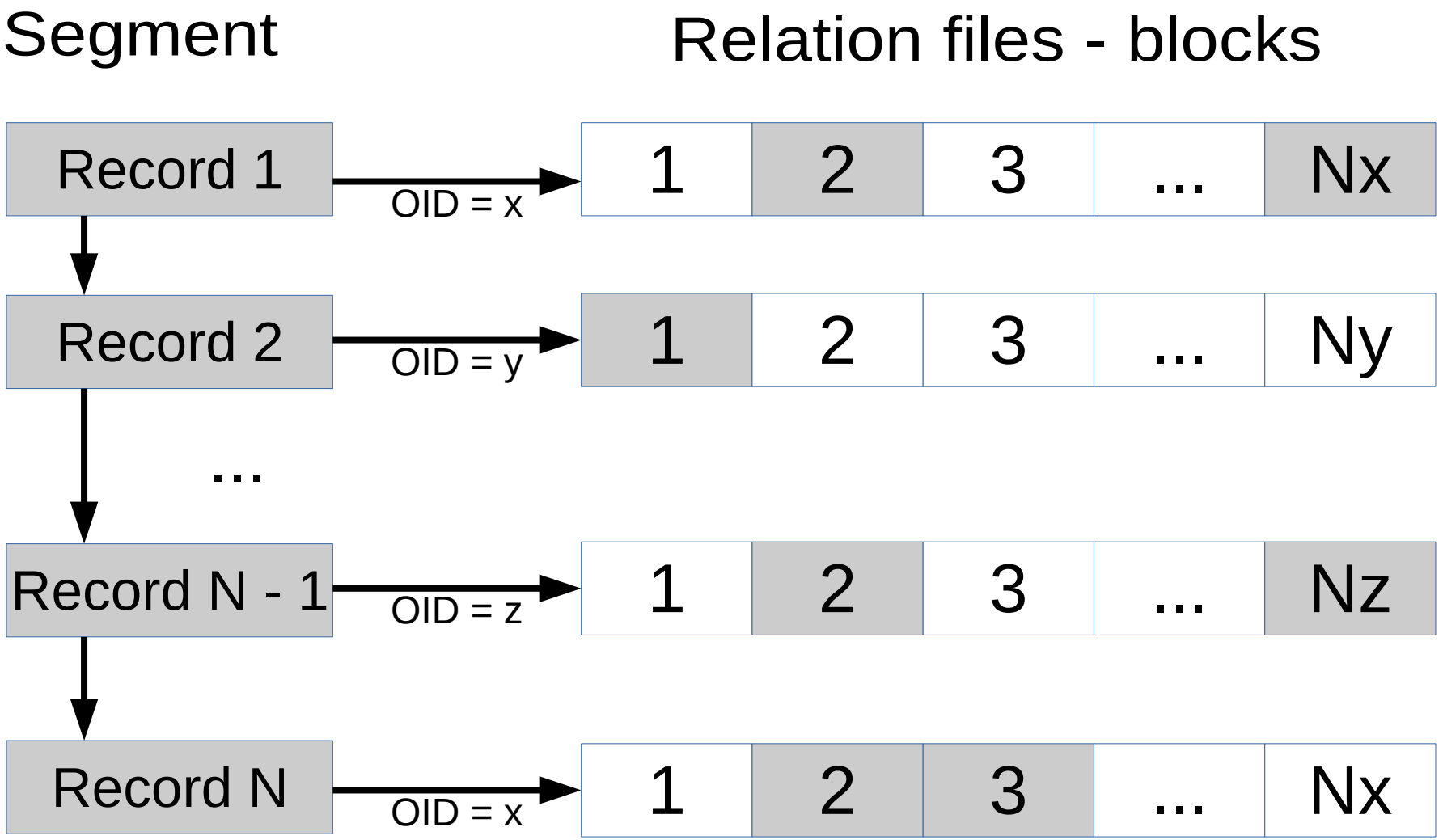
Existing solutions

- pg_rman
  - Differential backup
  - Scan each relation file and fetches modified blocks
  - Backups are smaller
  - Actually slow on large sets
- barman
- pitrery
- pg_bman
- Etc. Tell me!

PostgreSQL 9.5

- WAL refactoring
  - Track relation block changes in WAL records
  - No need to look at the record type
  - Generic approach
- pg_rewind
  - Integrated in 9.5
  - Uses similar logic
- Base for differential backup

WAL segments – block tracking



WAL segments – block mapping

- From WAL position (LSN) A to B

```
Relation X: {2, 3, Nx}
Relation Y: {1}
Relation Z: {2, Nz}
```

- LSN
  - WAL position, like 0/14EBDA0
- Segment (16MB by default)
  - **00000001**00000063**00000027**

Differential backup

- Last full backup taken
  - Uses pg_start_backup(), LSN X
  - Does backup, cp, tar, etc.
  - pg_stop_backup()
- Differential
  - Launches pg_start_backup(), LSN Y
  - Scans WAL segments from X to Y and gets mapping
  - Fetch modified blocks
  - pg_stop_backup()

Rebuilding full backup

- Determine last full backup
- Checks list of full backups up to wanted target
- Applies diff to relation data files
- Use backup_label of last diff backup
- Create recovery.conf

pg_arman

- Fork of pg_rman, largely simplified
  - ERROR layer simplified
  - Re-thinking of fancy options
  - Many code simplifications
- Full **and** differential (page-level) backup
- Removal of page holes
- Thanks, Yury Zhuravlev (Postgres Pro)!

pg_arman - 2

- Restrictions - hint-bit updates
  - wal_log_hints = on
  - Page checksums => initdb -k
- Applies diff backups stupidly in chain on a file base
- Does not support backup using stream (could be done)
- Backup taken on same host as Postgres instance
- PostgreSQL license
- Pet project:
  - https://github.com/michaelpq/pg_arman/
  - Has documentation!

Backup performance

- Data size ~ WAL segment quantity
  - Full backups preferable
  - Similar I/O read
- Data size >> WAL segment
  - Differential backup
  - May be costly if same blocks are always modified
- Important to leverage backup frequency
- Testing is important here!

pg_arman demonstration

Usage:
 pg_arman OPTION init
 pg_arman OPTION backup
 pg_arman OPTION restore
 pg_arman OPTION show [DATE]
 pg_arman OPTION validate [DATE]
 pg_arman OPTION delete DATE

Improvements

- Relation map
  - Used across multiple diff backups
  - Generated and rebuilt at each backup
- Acceleration of restore time (critical)
  - Multiple diff backup problem
  - Reuse relation map
  - Maximum load at backup time
- Stream mode
  - Superuser-based
  - File access functions
  - Replication protocol

Thanks!
Questions?